Tópicos Especiales en Telemática: Proyecto 4 Clústering de documentos a partir de métricas de similitud basado en Big Data

Daniel Hoyos-Ospina
Escuela de ingeniería
Pregrado en ing. de sistemas
Universidad EAFIT
Medellín, Colombia
Email: dhoyoso@eafit.edu.co

Daniela Serna-Escobar
Escuela de ingeniería
Pregrado en ing. de sistemas
Universidad EAFIT
Medellín, Colombia
Email: dsernae@eafit.edu.co

***Abstract*—Este documento plantea una solución al problema de agrupamiento (clústering) de un conjunto de documentos de texto no ordenados, utilizando el algoritmo k-means y la función de similitud para hallar la distancia entre ellos y crear clústers significativos y coherentes, desarrollada e implementada en Python y Apache Spark basandose en modelos de programación en BigData.**

Palabras clave:

Similitud, Clústering, K-means, Agrupamiento, Minería de Datos, BigData, Hadoop, Spark, YARN, HDFS.

1. Introducción

La agrupación de documentos es una técnica útil usada para grandes volúmenes de documentos de texto que están en internet como bibliotecas digitales, repositorios, artículos de blogs, correos electrónicos, buscadores de web, sistemas de recomendación que hacen parte de los problemas asociados a computación de alto rendimiento y presentan un desafío para grandes compañías (Google, Facebook, Amazon, Netflix, Spotify, entre otros) pues la información debe ser organizada de forma correcta y automática en colecciones coherentes y eficaces, más ahora cuando los datos inundan los negocios cada día y deben analizarse para que sean útiles, que conduzcan a mejores decisiones y movimientos de negocio estratégicos.

Las métricas de comparación pueden variar según su propósito, por ejemplo en artículos de investigación dos documentos se consideran similares si comparten los mismos temas, pero para motores de búsqueda se compara según el tipo de información que presenta, sin embargo, para esta solución se concibe la similitud en términos de distancia y existen diferentes funciones de similitud, entre los mas usados están la distancia Euclidiana, Coseno, Jaccard y Pearson.

2. Marco Teórico (Descripción del problema)

2.1. Representación de documentos

Se tiene un conjunto de documentos de texto grande o un dataset, el cual debe presentar algunas relaciones sintácticas y semánticas entre los documentos, es decir, que hablen del mismo tema o pertenezcan al mismo género. Para esto se usó la base de textos de Gutenberg. Las palabras aparecen de forma independiente y sin importar el orden, se determina el peso de cada termino, es decir su frecuencia de aparición en los documentos, lo que significa que las palabras que aparecen con mayor frecuencia son las que mejor describen la similitud para los documentos. Pero existen palabras que aparecen en repetidas ocasiones y que no representan los textos o no son importantes, por ejemplo, conectores, artículos, pronombres, preposiciones etc. Que son filtradas antes del procesamiento de datos en lenguaje natural para garantizar que los documentos si sean comparados correctamente.

2.2. Desarrollo

Para comenzar con el algoritmo de agrupamiento y clustering, se requiere una estructura donde se agrupen los documentos, generalmente esta es una matriz td-idf. Esta matriz se compone de dos estadísticos: El primero, es la frecuencia de una palabra dentro del documento o tf (Term Frecuency), esta puede realizarse de varias maneras, la más común es un conteo crudo de las veces que se encuentra un término en el documento, también se puede realizar una frecuencia relativa del término dentro del documento, incluso se puede comparar la frecuencia cruda de cada palabra con la frecuencia más alta, entre otros.

El segundo estadístico consiste en que tan diferente es el termino dentro de todos los documentos, o frecuencia inversa de documento (Inverse Document Frecuency por sus siglas en inglés); Este usa la siguiente formula:

$$\mathrm{idf}(t, D) = \log \frac{N}{1 + |\{d \in D : t \in d\}|}$$

Figure 1. $https://en.wikipedia.org/wiki/Tf-idf$

En donde:

- idf(t,D) representa la frecuencia inversa del trmino t en el conjunto de documentos D.
- N representa el número total de documentos.
- 1+$|d \in D : t \in d|$ representa el número de documentos d en los cuales está el termino t. Se le suma uno para evitar la división por cero en caso de que la palabra no exista

El proposito del paso idf establece que las palabras más frecuentes en todos los documentos resultarán siendo las menos importantes y las no tan frecuentes serán las más representativas.

Este par de estadísticos se iteran por cada palabra y por cada documento, generando la matriz, donde las filas representan los documentos y cada columna representa cada termino. Algunas veces la cantidad de términos es excesiva y se vuelve muy complejo el manejo de la matriz, por lo que se pueden realizar heurísticas para la reducción de los términos, por ejemplo, es muy usual remover esas palabras que son muy frecuentes, o las que no se repitan más de 1 vez, o incluso acotar a ambos lados, todo esto depende del conjunto de documentos por lo que se debe conocer con anterioridad para establecer un buen criterio de acotamiento.

El resultado será no solo determinar la diferencia entre los documentos sino encontrar los distintos clústeres de los subgrupos dependiendo de la similitud entre ellos y se presentará una solución que evidencie la reducción de tiempos entre el proyecto HPC y BigData y muestre como las estrategias, herramientas, infraestructura y tecnologías permiten resolver adecuadamente problemas.

2.3. Algoritmo de clústering

El algoritmo K-means trabaja con medidas de distancia que básicamente apuntan a minimizar las distancias dentro del clúster. Por lo tanto, las medidas de similitud no encajan directamente en el algoritmo, porque los valores más pequeños indican desemejanza. La distancia euclidiana y la divergencia KL promediada son medidas de distancia, mientras que la similitud del coseno, el coeficiente de Jaccard y el coeficiente de Pearson son medidas de similitud. Se tuvo que aplicar una transformación simple para convertir la medida de similitud en valores de distancia.

Dado un conjunto de objetos D y un número preestablecido de clústeres K elegidos al azar. Cada k es un centroide (centro geométrico) de un clúster y los objetos restantes se asignan al clúster representado por el centroide más cercano o más similar. A continuación, se vuelven a calcular los centroides nuevos para cada grupo y a su vez todos los nuevos documentos en base a los nuevos centroides. Además, se saca un grupo de documentos sobrantes, es decir, aquellos que no tienen relación con ningún centroide.

Las soluciones de agrupamiento generadas son óptimas para el conjunto de datos dato y las semillas iniciales, las diferentes elecciones de conjuntos y clústeres iniciales pueden dar lugar a particiones finales muy diferentes y por ende a tiempos de ejecución variantes. Sin embargo, se utilizó el algoritmo básico de K-means porque optimizar el agrupamiento no es el enfoque de este proyecto.

Los resultados son óptimos cuando se minimiza la distancia intra-clústern (cohesión) y se maximiza la distancia inter-clúster (separación), pero este algoritmo no garantiza que los centroides finales obtenidos sean los mejores.

Este algoritmo se considera NP-Hard lo que significa que es computacionalmente difícil de abarcar, esta clasificacin se deriva desde el algoritmo de similitud siendo simplemente aproximaciones, luego está la selección del k, y por último está la definición de los centroides; Estas consideraciones de precisión no están dentro del alcance del proyecto por lo que se realizan heurísticas muy básicas para su determinación.

3. Análisis y diseño mediante Analítica de Datos (ETL-Procesamiento-Aplicación), incluyendo: algoritmos, estructuras de datos, metadatos de datasets y rendimiento analítico de la solución.

Extraer, transformar y cargar (ETL) es el proceso que permite a las organizaciones mover datos desde múltiples fuentes, reformatearlos, limpiarlos, y cargarlos en otra base de datos, data mart, o data warehouse para analizar, o en otro sistema operacional para apoyar un proceso de negocio. En nuestro caso este es usado para cargarlos en el formato necesario para la ejecución del procesamiento paralelo del algoritmo K-Means implementado en Apache Spark.

3.1. Extracción

La primera parte del proceso ETL consiste en extraer los datos desde el sistema de origen. En nuestro caso el origen de los datos es el file system, puntualmente del Hadoop File System representados en formato de ficheros planos. La extracción se encarga de convertir los datos a un formato de destino preparado para iniciar el proceso de transformación. En nuestro caso el formato de destino es una Dataframe de spark que maneja estructuras tipo Tablas Hive columnares, la tabla tendrá dos columnas; una contendrá la dirección de cada archivo y la otra contendrá el texto de cada archivo.

3.2. Transformación

La fase de transformación aplica una serie de funciones sobre los datos extraídos para convertirlos en datos que serán cargados. El k-means implementado en la librería mllib de Spark no recibe textos como tal si no el peso de las palabras que contiene. Este se extrae a partir de tokenizar el texto (separar las palabras para ser tratadas individualmente), quitar las stop words, listar sus frecuencias a partir del método hashing TF, y por ultimo realizar un estadístico conocido como IDF (Inverse Document Frequency) encargado de quitar importancia a las palabras que se repiten en muchos documentos ya que estas no representan factores de diferenciación entre ellos.

El resultado es una matriz i x j en donde i son los documentos, n las palabras que se toman en cuenta para el algoritmo de comparación y cada posición [i][j] representa el peso de la palabra i en el documento j, esta matriz es el principal parámetro del algoritmo KMeans.

3.3. Carga (Load)

La fase de carga es el momento en el cual los datos de la fase anterior (transformación) son cargados en el sistema de destino.

En este caso el algoritmo K-Means recibe el K (número de clúster deseado) y la matriz generada en el paso anterior. El algoritmo crea un modelo a partir de esta matriz que identifica y genera los centros de cada clúster para futuras predicciones.

3.4. Procesamiento

El procesamiento de los datos de carga se realiza en primera instancia en la creación del modelo K-Means. Este modelo es quien identifica los centros de los clústeres que son los que usa posteriormente para clasificar los documentos en determinado clúster. Por ahora solo se tiene el modelo, y los centros de clúster, es decir, aún no se han identificado a que clústeres pertenecen los documentos. Este paso de definir el modelo es el más importante ya que afecta directamente el resultado de cualquier predicción que el KMeans vaya a realizar.

3.5. Aplicación

Para obtener los clústeres de los documentos que deseamos procesar es necesario entonces usar la función de predicción del modelo KMeans llamada transform que al pasarle la tabla de textos generada en la fase de extracción devuelve una columna nueva que dice a qué clúster pertenece el texto de esa fila. Entonces usando esta función podemos determinar los K clústeres y saber que documentos hacen parte estos.

4. Implementación

Esta implementación de clústering de documentos está hecha usando el nuevo DataFrame API de Apache Spark y Python (pyspark), tomando como base que la diferencia entre documentos radica principalmente en el peso que poseen las palabras dentro de cada documento en particular. En este pensamiento el peso de las palabras se calcula a partir de los términos únicos que un documento proporciona al conjunto de todos los documentos además de los términos comunes, lo que le permite agruparse con documentos similares.
El pensamiento detrás de este enfoque es que si hay algunos artículos que tienen un conjunto común de palabras, es probable que el artículo con más palabras únicas cubra el tema con mayor detalle que los otros. Además, hay un límite no determinista para estas palabras únicas, más allá del cual el algoritmo de agrupamiento lo expulsará del clúster. Este pensamiento se valida aún más por la forma en que los motores de búsqueda típicos utilizan un índice documentado invertido para encontrar los documentos más relevantes para un término de búsqueda determinado.
La implementación comienza estableciendo el contexto de Spark, este contexto es el punto de entrada a todo el ambiente del Hadoop File System, usando el contexto podemos leer archivos de manera individual con la función textFile o como lo hacemos en nuestro, leer un conjunto de documentos con la función wholeTextFiles. Con el fin de probar la efectividad del enfoque de agrupamiento, seleccioné datos de diferentes fuentes sobre tres autores diferentes: Abraham Lincoln, Charles Dickens y Bram Stocker. Estos datos son leídos como se dijo en el párrafo anterior y una vez abiertos creamos un DataFrame por medio de SparkSession, objeto de la librería sql de spark para el manejo de DataFrames y DataSets; cada punto de datos está representado por una tabla con el siguiente formato similar a una tabla de base de datos:

- Path: Esta es la ruta de origen del documento por ejemplo:
  $hdfs:///user/dataset/Abraham_lincoln.txt$
- Delimitador: El — es el delimitador.
- Text: Es el contenido extraído del origen. Simplemente es el texto plano de cada documento.

Por ejemplo:

path	text
hdfs://hdplabmast...	LINCOLN LETTERS...
hdfs://hdplabmast...	Lincoln's First...
hdfs://hdplabmast...	Lincoln's Get.....
hdfs://hdplabmast...	Lincoln's Firs....
hdfs://hdplabmast...	Lincoln's Secon...

Esta tabla será el insumo principal para empezar el proceso de transformación necesario antes de instanciar el

KMeans y usarlo.

Todo el proceso de transformación es descrito a través de un ML pipeline el cual permite aplicar un conjunto de transformaciones a uno o varios Dataframes y obtener el resultado.

Nuestro pipeline tiene las siguientes fases:

4.1. Tokenize

En primer lugar debemos coger la tabla inicial y transformar la columna texto de tipo string en una columna de tokens de tipo String array en donde cada posición del arreglo contendrá cada palabra del documento. El marco de datos tiene ”path” para la fuente (la clave RDD emparejada) y ”tokens” para las palabras (el valor RDD emparejado).

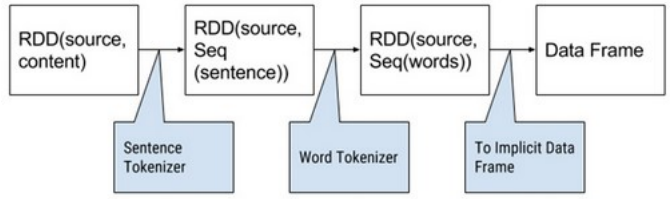


Figure 2. $https : //www.3pillarglobal.com/insights/topic - clusters - tf - idf - vectorization - using - apache - spark$

El marco de datos queda así:

path	text	tokens
hdfs://hdplabmast...	LINCOLN LETTERS...	[, , lincoln, let...
hdfs://hdplabmast...	Lincoln's First...	[, , lincoln's, f...
hdfs://hdplabmast...	Lincoln's Get...	[, , , , lincoln'...
hdfs://hdplabmast...	Longman's Engli...	[, , longman's, e...

Despúes de tokenize se crea otra columna con la misma estructura de tokens, en esta columna se almacenarán los tokens procesados sin stop words por medio de la clase StopWordsRemover, la cúal tiene la opción de cargar el idioma de los documentos.

4.2. Hashing TF

Esta transformación es proporcionada por la biblioteca de machine learning de Spark y transforma cada palabra en su hash. La entrada de esta transformación es una secuencia de términos y la salida es un Sparse vector. Un vector disperso almacena los índices con dimensiones activas y los valores en cada dimensión. En este caso, cada índice es el resultado de la función de hashing para el término y el valor en el índice es la frecuencia del término en el nodo de datos (documento). Además de esto la función de hashing recibe un parámetro que limita la cantidad de índices que va a manejar. Esta transformación usa como entrada la columna de tokens sin stop words y como salida la columna rawFeatures dando como resultado un marco de datos así:

path	stopWordsRemovedtokens	rawFeatures
hdfs://hdplabmast...	[, , lincoln, let...	(2000,[4,9,12,24,...
hdfs://hdplabmast...	[, , lincoln's, f...	(2000,[0,1,4,5,8,...
hdfs://hdplabmast...	[, , , , lincoln'...	(2000,[25,61,66,8...
hdfs://hdplabmast...	[, , longman's, e...	(2000,[0,1,2,3,4,...
hdfs://hdplabmast...	[, , lincoln's, s...	(2000,[0,6,17,25,...

El 2000 representa que solo se tendrán en cuenta los 2000 índices más relevantes.

4.3. IDF Transform

Esta transformación suministrada por Spark calcula la frecuencia inversa del documento de cada término. El puntaje TF-IDF para un término que es único en todos los cuerpos de texto de los documentos es más alto que un término que es común. Por lo tanto, las dimensiones inferiores de un vector TF-IDF para un documento representan su parte de términos comunes en el conjunto total de cuerpos de texto, mientras que las dimensiones más altas representan los términos únicos.
Esta transformación usa como entrada la columna de rawFeature y como salida la columna features, además recibe un parámetro llamado minDocFrequency el cual representa la cantidad mínima de apariciones en documentos diferentes para tener en cuenta esa palabra, dando como resultado un marco de datos así:

path	rawFeatures	features
hdfs://hdplabmast...	(2000,[4,9,12,24,...	(2000,[4,9,12,24,...
hdfs://hdplabmast...	(2000,[0,1,4,5,8,...	(2000,[0,1,4,5,8,...
hdfs://hdplabmast...	(2000,[25,61,66,8...	(2000,[25,61,66,8...
hdfs://hdplabmast...	(2000,[0,1,2,3,4,...	(2000,[0,1,2,3,4,...
hdfs://hdplabmast...	(2000,[0,6,17,25,...	(2000,[0,6,17,25,...

Aparentemente se ven iguales pero es porque la impresión no permite mostrar todos los features y allí si se nota la diferencia.

4.4. KMeans

El último paso del pipeline es el KMeans el cual recibe como parámetro de entrenamiento el resultado de la columna features (IDF) el cual en realidad viene a ser una matriz pues cada posición de la columna tiene un arreglo de features en teoría viene siendo un arreglo de arreglos en donde cada fila es relacionada al documento, cada columna a la palabra en ese documento y la cada posición es el peso de la palabra de la columna en el documento de la fila.

Por medio del método fit del pipeline podemos crear el modelo para el KMeans; Este método recibe como parámetro el DataFrame inicial para el entrenamiento del

KMeans, es decir, todos los pasos realizados anteriormente, fueron exclusivos para el entrenamiento del KMeans, lo que nos deja como resultado un modelo con solo los centros de los establecidos.

Luego pasamos a predecir el nuevo conjunto de datos que para el caso de esta implementación será el mismo al de entrenamiento, esto arrojará como resultado los grupos explícitos que fuerón conformados para hallar los centros; Esto se realiza por medio de la función transform del modelo.

Para imprimir los resultados se hace un select del nombre de los documentos y de la predicción que arroja el KMeans, se agrupa el resultado por predicción, se hace collect a los documentos en una lista y se guarda la tabla en formato JSON así:

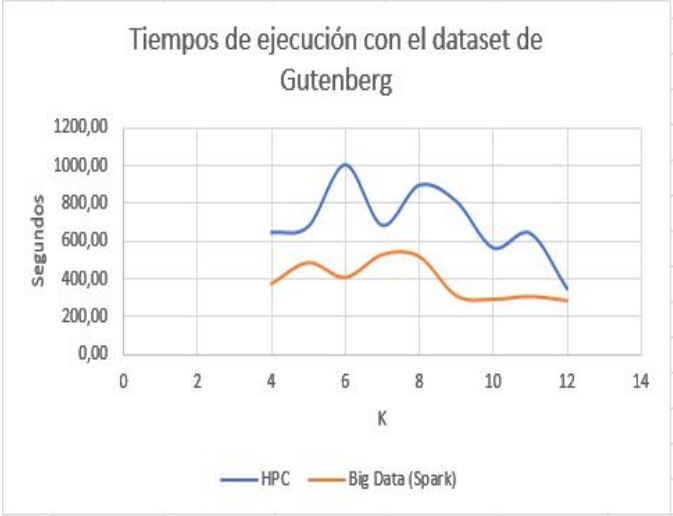

```
{
"prediction":1,
"cluster_docs_list":[
"Alfred Russel Wallace___Contributions.txt",
"Alfred Russel Wallace___Darwinism.txt",
"Alfred Russel Malay Archipelago Volume 1.txt"]
}
{
"prediction":6,
"cluster_docs_list":[
"Frank Richard Stockton___The Dusantes.txt",
"Frank Richard Stockton___The House of Martha.txt"
]
}
```


5. Análisis de resultados (HPC vs Big Data, gráficos de aceleración y distintas ejecuciones)

Para probar el código se uso un sub set de documentos del Project Gutenberg que ofrece más de 54,000 eBooks gratuitos. las obras más antiguas para las cuales el copyright ha expirado. el dataset que usaremos es nada más una pequea parte de este que consta de 461 eBooks.

Al correr las implementaciones en HPC usando MPI4py y Big data usando Pyspark variando la K en el dataset de Gutenberg en espaol, se observa que el uso de Spark mejora el tiempo de ejecución promedio en el orden de 2,048570985 veces más velóz que HPC con el dataset de Gutenberg como refleja las siguientes ilustraciones.

Tiempos de ejecución con el dataset de Gutenberg		
K	tiempo HPC (segs)	Tiempo Big Data (segs)
4	647,40	376,48
5	679,87	485,65
6	1008,48	409,43
7	685,55	527,33
8	899,38	517,20
9	814,65	315,02
10	565,20	296,82
11	643,30	311,42
12	348,22	289,73



En ambos casos las pruebas se realizaron usando 10 núcleos.

Se logró evidenciar que las aceleraciones no fueron uniformes en todas las ejecuciones, a pesar de ejecutarse bajo las mismas circunstancias y usarse el mismo K, el mismo código y la misma cantidad de núcleos.

6. Conclusiones

La elaboración de los proyectos permitió ampliar los conocimientos relacionados con big data, clústering y paralelismo, que sin duda son muy importantes en el futuro de la computación.

Se logró el objetivo del proyecto (uso de paralelismo) gracias al uso ordenado y planeado de la metodología ETL-Procesamiento-aplicación.

El uso de la programación paralela y el big data combinados con estrategias y herramientas, son de vital importancia para resolver problemas que sin ellos no podrían ser solucionados o requerirían recursos computacionales mucho mayores.

Se logró reducir tiempos de ejecución mediante estrategias,

tecnologías, infraestructura, herramientas y algoritmos de la libreria Spark.

Se implementó un algoritmo de clústering (k-means) para sortear los retos que presenta hoy la minería de texto.

Se concluyó que la desvación estandar de los tiempos de ejecución en HPC usando MPI4Py varian más que en Big Data usando pyspark mllib.

El uso de Spark para la implementación de clústering de documentos con KMeans resultó hasta un 100% más velóz.
La herramienta de Spark en la implementación de clústering de documentos con KMeans tuvo resultados más precisos y lógicos.

Referencias

Fernando Berzal. (2011). *Clustering.* Tomado de $http://elvex.ugr.es/decsai/intelligent/slides/dm/D3\%20Clustering.pdf$

Anna Huang. *Similarity Measures for Text Document Clustering.* Tomado de $http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.332.4480\&rep=rep1\&type=pdf$

Christopher D. Manning, Prabhakar Raghavan, and Hinrich Schtze. (2009). *Introduction to Information Retrieval.* Tomado de $https://nlp.stanford.edu/IR–book/pdf/irbookonlinereading.pdf$

Anand Kumar Gupa and Neetu Sardana. (2015). *Significance of Clustering Coefficient.* Tomado de $http://ezproxy.eafit.edu.co:2214/xpls/icp.jsp?arnumber=7346726$

Programación Paralela. Tomado de $https://www.cs.buap.mx/\ mtovar/doc/ProgConc/ProgramacionParalela.pdf$

Gutenberg dataset. Tomado de $http://www.gutenberg.org/$

Data clustering using Apache Spark. Tomado de $https://dzone.com/articles/cluster-analysis-using-apache-spark-exploring-colo$

Using pyspark KMeans for Real World Clustering Problems. Tomado de $http://www.jianshu.com/p/84d6b6b7874a$

KMeans Clustering in Spark. Tomado de $http://machinelearningandspark.blogspot.com.co/2016/05/k-means-clustering-in-spark.html$

KMeans Clustering with Map Reduce. Tomado de $http://codingwiththomas.blogspot.com.co/2011/05/k-means-clustering-with-mapreduce.html$